



INVESTOR IN PEOPLE

The Patent Office
Concept House
Cardiff Road
Newport
South Wales
NP10 8QQ

I, the undersigned, being an officer duly authorised in accordance with Section 74(1) and (4) of the Deregulation & Contracting Out Act 1994, to sign and issue certificates on behalf of the Comptroller-General, hereby certify that annexed hereto is a true copy of the documents as originally filed in connection with the patent application identified therein.

In accordance with the Patents (Companies Re-registration) Rules 1982, if a company named in this certificate and any accompanying documents has re-registered under the Companies Act 1980 with the same name as that with which it was registered immediately before re-registration save for the substitution as, or inclusion as, the last part of the name of the words "public limited company" or their equivalents in Welsh, references to the name of the company in this certificate and any accompanying documents shall be treated as references to the name with which it is so re-registered.

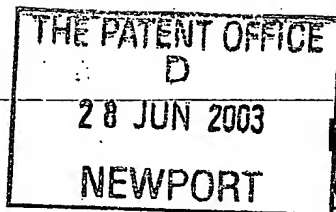
In accordance with the rules, the words "public limited company" may be replaced by p.l.c., plc, P.L.C. or PLC.

Re-registration under the Companies Act does not constitute a new legal entity but merely subjects the company to certain additional company law rules.

Signed

Dated

1 September 2003



The
Patent
Office

1/77

Patents Act 1977
Rule 16

Request for grant of a patent

The Patent Office
Concept House
Cardiff Road
Newport
South Wales NP10 8QQ

1.	Your reference	GB920030026GB1		
2.	Patent application number (The Patent Office will fill in this part)	0315193.3		28 JUN 2003
3.	Full name, address and postcode of the or of each applicant (underline all surnames)	INTERNATIONAL BUSINESS MACHINES CORPORATION Armonk New York 10504 United States of America		
	Patents ADP number (if you know it)	519637001		
	If the applicant is a corporate body, give the country/state of its incorporation	State of New York United States of America		
4.	Title of the invention	SYSTEM AND METHOD FOR USER INTERFACE AUTOMATION		
5.	Name of your agent (if you have one)	D P Litherland		
	"Address for Service" in the United Kingdom to which all correspondence should be sent (including the postcode)	IBM United Kingdom Limited Intellectual Property Department Hursley Park Winchester Hampshire SO21 2JN		
	Patents ADP number (if you know it)	919006		
6.	If you are declaring priority from one or more earlier patent applications, give the country and the date of filing of the or of each of these earlier applications and (if you know it) the or each application number	Country	Priority App No (if you know it)	Date of filing (day/month/year)
7.	If this application is divided or otherwise derived from an earlier UK application, give the number and the filing date or the earlier application	No of earlier application	Date of filing (day/month/year)	

8. ☒ Is a statement of inventorship and of right to grant of a patent required in support of this request? (Answer 'Yes' if:
a) any applicant named in part 3 is not an inventor, or
b) there is an inventor who is not named as an applicant, or
c) any named applicant is a corporate body.) Yes


9. Enter the number of sheets for any of the following items you are filing with this form. Do not count copies of the same document

Continuation sheets of this form

Description 17

Claim(s) 3

Abstract 1

Drawing(s) 6 

10. If you are also filing any of the following, state how many against each item.

Priority documents

Translations of priority documents

Statement of inventorship and right to grant of a patent (Patents Form 7/77) 5

Request for preliminary examination and search (Patents Form 9/77)

Request for substantive examination (Patents Form 10/77)

Any other documents (please specify)

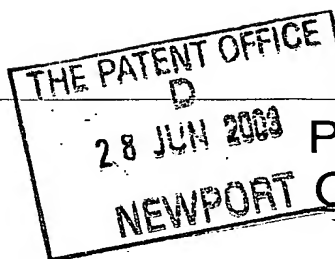
11. I/We request the grant of a patent on the basis of this application



Signature

27 June 2003
Date

12. Name and daytime telephone number of person to contact in the United Kingdom D P Litherland
01962 816303




The
Patent
Office

7/77

Patents Act 1977
Rule 15

Statement of inventorship and of right to grant of a patent

The Patent Office
Concept House
Cardiff Road
Newport
South Wales NP10 8QQ

1.	Your reference	GB920030026GB1	
2.	Patent application number (if you know it)	0315193.3	28 JUN 2003
3.	Full name of the or of each applicant	INTERNATIONAL BUSINESS MACHINES CORPORATION	
4.	Title of invention	SYSTEM AND METHOD FOR USER INTERFACE AUTOMATION	
5.	State how the applicant(s) derived the right from the inventor(s) to be granted a patent	By employment and by agreement	
6.	How many, if any, additional Patents Forms 7/77 are attached to this form?	1	
7.	I/We believe that the person(s) named over the page (and on any extra copies of this form) is/are the inventor(s) of the invention which the above patent application relates to.		
	 Signature		27 June 2003 Date
8.	Name and daytime telephone number of person to contact in the United Kingdom	D P Litherland Tel: 01962 816303	

Enter full names, addresses and postcodes of the
inventors in the boxes and underline the surnames

Simon PIZZOLI
(Resident of Ireland)
c/o IBM United Kingdom Limited
Intellectual Property Law
Hursley Park
Winchester
Hampshire SO21 2JN
England

Patents ADP number (if known)

866339 5001

Hugh P SMYTH
(Resident of Ireland)
c/o IBM United Kingdom Limited
Intellectual Property Law
Hursley Park
Winchester
Hampshire SO21 2JN
England

Patents ADP number (if known)

866340 3001

If there are more than three inventors, please write their names and addresses on the back of another Patents Form 7/77 and attach it to this form

Patrick Joseph O'SULLIVAN
(Resident of Ireland)
c/o IBM United Kingdom Limited
Intellectual Property Law
Hursley Park
Winchester
Hampshire SO21 2JN
England

Patents ADP number (if known)

866341 1001

REMINDER

Hav you signed th form?

Enter the full names, addresses and postcodes of the inventors in the boxes and underline the surnames

Mark E WALLACE
(Resident of Ireland)
c/o IBM United Kingdom Limited
Intellectual Property Law
Hursley Park
Winchester
Hampshire SO21 2JN
England

Patents ADP number (if known)

8663437001

Patents ADP number (if known)

If there are more than three inventors, please write their names and addresses on the back of another Patents Form 7/77 and attach it to this form

Patents ADP number (if known)

REMINDER

Have you signed the form?

SYSTEM AND METHOD FOR USER INTERFACE AUTOMATION

Field of the Invention

5 This invention relates to the general field of user interface automation, and particularly (though not exclusively) runtime graphical user interface automation.

Background of the Invention

10 User interface automation involves the development of technical test scripts which will execute and which will allow an application to be navigated and tested in an automated way, i.e., with no user intervention. If the automation scripts are well written then errors/bugs will be
15 automatically recorded so that an engineer can acknowledge and address these bugs when the automation scripts have run to completion. Once the bugs are fixed it is a common practice to execute the automation scripts again, thereby proving that the application being tested is bug free.

20 Automation scripts are commonly written so as to 1) significantly speed up the software testing process, 2) allow automation to run when engineers are away from the office (e.g., at night), and 3) develop test cases which are complete and comprehensive. Automation is therefore a
25 useful tool/methodology for reducing costs of quality engineering and testing, as it speeds up the same activity that would otherwise be done manually.

30 One of the main problems/drawbacks with automation scripts is that they need to be heavily modified so as to execute on translated versions of an application (and it is common knowledge that testing on translated
35 versions of the application accounts for over 80% of a corporation's worldwide testing effort). This process of modification (so that the automation can be run against translated versions of an application) is time consuming and error prone in a number of ways. Once the automation is
40 developed it traditionally needs to be "re-engineered" to accommodate the characteristics of different language versions of the application. In particular, references that the automation makes to textual components on screen are invalid when the automation is executed on a translated version of the same application. Re-engineering implies fundamental structural changes to the automation to allow it to run on the translated versions of
the application. Subsequent testing, revision and re-testing of the automation is also time consuming. Finally, when the automation is modified to cater for a subsequent version of the application (e.g. Version 2) the cycle of re-engineering, revision and re-testing needs to happen again.

Therefore, the current industry methodology for developing automation results in a process which is time consuming and expensive when translated/language versions of the same application are considered. Because of this, few companies develop automation to run on translated versions of the application (i.e., it is generally written and designed for English versions of the application).

From U.S. patent no. 5,475,843 there are known a system and methods for automated program testing which require predetermined, systematic and complex interception of the running application at run time. The level of interception is intrusive, and has the disadvantage of creating a high probability of compromising the natural execution of the application. In Quality Engineering terms, methods of automation are gauged on their effectiveness, and one measure is the level of interference for the running application being tested. The methods used in this patent rely on an interception-based methodology or system, rather than allowing the application to run unhindered.

A second drawback to the methodology used in this patent is that it cannot intercept and control the resolution and management of complex user interface control types (such as 'InfoBox' panels in IBM® Notes™ and Smartsuite™ products, or a number of the more complex user interface classes in J2EE - Java 2 Platform, Enterprise Edition).

Further, in this patent's methodology the application is centric, i.e., the interception of all application messages, events (keyboard, mouse, system), states, etc., is a pre-requisite to success. Such a model once again interferes with the natural execution of the application. This gives rise to quality and engineering considerations which can undermine the true value of the automation being run.

Finally, the methodology used in this patent encourages a level of abstraction which can potentially take some degree of control from the script writer. In this regard the script writer is somewhat dependent on the methodology and characteristics of the this patent's interception and event handling engine. This limits the degree of control that the script writer can have in the this patent's natural test abstraction development methodology.

In summary, this patent's automated testing solution behaves in a way which encourages interference and interception of the running application. This patent relies on the specific interception of the running application to achieve its goals, and relies on the need to write ATUs (Application Translation Units).

A need therefore exists for a system and method for language-neutral runtime user interface automation wherein the abovementioned disadvantage(s) may be alleviated.

Statement of Invention

In accordance with a first aspect of the present invention there is provided a system for user interface automation as claimed in claim 1.

In accordance with a second aspect of the present invention there is provided a method for user interface automation as claimed in claim 10.

The present invention allows the above problems to be solved and provides a framework for a solution which will allow automation scripts to be developed once and re-used unmodified against translated versions of the same application.

In a preferred embodiment, the present invention allows run-time mapping of user interface components of an automation script using an original-language (e.g., English) automation script as the reference; this has great potential because it does not attach any extra burden on the script developer and requires little extra from the user - other than providing an English reference file.

Brief Description of the Drawing(s)

One system and method for language-neutral runtime user interface automation incorporating the present invention will now be described, by way of example only, with reference to the accompanying drawing(s), in which:

FIG. 1 shows a block schematic illustration of components involved in prior art execution of automated testing;

FIG. 2 shows a block schematic illustration of how steps of a 'Log On' Test Script relate to user interface elements in the method of FIG. 1;

FIG. 3 shows a block schematic illustration of possible failure of the 'Log On' Test Script of FIG. 2 when the application to be tested is translated;

FIG. 4 shows a block schematic illustration of a test scheme, in accordance with a preferred embodiment of the present invention, in which translation difficulties are obviated;

BEST AVAILABLE COPY

FIG. 5 shows a block schematic illustration of a possible sequence of steps for implementing intercept and execution of a user interface action in the test scheme of FIG. 4; and

FIG. 6 shows a block schematic illustration of steps involved in the intercept and execution implementation of FIG. 5.

Description of Preferred Embodiment(s)

A typical test automation tool provides the following features:

- Scripting language which is used to describe the logic of the test case, e.g., including basic flow control semantics.
- A library of functions which will simulate user actions on the application graphical user interface, e.g., a button click.
- Result reporting library which will record test script execution, passes and failures.
- Test script interpreter which will execute the test cases.

FIG. 1 illustrates the components involved in the execution of automated testing: a test script 110, a test script interpreter 120, a target application 130 and test script results 140

In the following example a section of the test script such as 110 will log a user on. A pictorial illustration of such a test script section is shown in FIG. 2, in which the test script 210 is associated with an interactive 'Log On' dialog 220 displayed on a user display (not shown). The user interacts with the displayed dialog 220 by entry of text via a keyboard (not shown) and movement of a display pointer (not shown) and clicking of a selection device such as a mouse (not shown) in conventional manner. The script includes a line 'EditSet Text ("User name", "John Doe")' which is associated (as shown by arrow 230) with a 'User name' text input field in the dialog 220, a line 'EditSet Text ("Password", "password")' which is associated (as shown by arrow 240) with a 'Password' text input field in the dialog 220, and a line 'ButtonClick ("Log On")' which is associated (as shown by arrow 250) with a 'Log On' button (or a 'Cancel' button) in the dialog 220. To achieve the desired result, the user must enter the text 'John Doe' in the 'User name' field, must enter the text 'password' in the 'Password' field, and must click 'Log On' button.

A test tool provides functions that allow the automation of the manual user steps of

(1) Selecting the 'User name' edit control

- (2) Typing a user's name
- (3) Selecting the 'Password' edit control
- (4) Typing a user's password
- (5) Clicking the 'Log On' button

5

In the above example the edit controls and button are identified by the text label associated with them, which would be intuitive for the test script author.

10

Now consider what happens when the application to be tested is translated (in this example into German), as shown in FIG. 3.

15

Because the text labels have been translated, the translated 'Anmeldung' dialog 320 (corresponding to the dialog 220, shown in this figure - for comparison only - by dashed line) is displayed to the user by the translated application. As shown, the translated 'Anmeldung' dialog 320 comprises a text input field labeled 'Benutzername', a text input field labeled 'Kennwort', and buttons labeled 'Anmeldung' and 'Abbrechen'. Consequently, it will be understood that the test script will fail (as indicated by the crossed arrow 330) at the first step 'EditSet Text ("User name", "John Doe")' described above and will report an error.

20

25

To prevent the failure of the test script, the following testing scheme is employed to provide a mechanism to automatically translate the text string used to identify the control being manipulated. FIG. 4 illustrates the modified system 400 for the execution of test automation which takes into account the translation of the target application, i.e., the application being tested. As shown, the system 400 includes test script 410, script translation engine 420 (which will be described in greater detail below) and supporting libraries of source resources 430 and target resources 440, script interpreter 450, target application 460 and test results 470.

30

35

As will be described in greater detail below, the Script Translation Engine is a new component, which performs the following functions:

40

- Intercept calls to functions which simulate user actions on the application,
- Retrieve the resource identifier associated with the text string which identifies the target control,
- Retrieve the translated string associated with the resource identifier, and

- Call the original function which simulates the user action but with the translated string.

The Script Translation Engine 420 includes a Message Translation Engine (MTE) 422 which is responsible for supplying the translated text for the automation script's run time execution, and an Intelligent Text Locator (ITL) 424 which is responsible for selectively supplying the appropriate text to the automation script's run time execution. For example, the ITL 424 should not parse the externalised resources in a generic way. It should source these in the context of the instruction being applied. If the script engine is processing a button (e.g. 'press OK', 'press Cancel') then the ITL must provide the relevant dialog text. The MTE 422 differs from the ITL 424 in a number of ways, a key one of which is that the MTE could search for a translation to a textual object, but the search may get satisfied with an incorrect translation. This can typically happen when there are multiple instances of a textual objects (i.e. a word or a sentence can appear in multiple places). It is the job of the ITL to ensure that during a search the correct translation is returned and that it respects the nature and type of control that the textual object belongs to - as opposed to simply taking the first match from the MTE:

When the call for a user action is intercepted the control type and translatable text must be identified. This can be inferred from the name of the function being called, e.g., 'ButtonClick' refers to push buttons, and the signature of the function (i.e., 'ButtonClick' takes a single argument which is the text label of the button to be clicked).

There are numerous examples of how the calls could be intercepted but the goal is to do this without requiring any test script alteration. For example, as shown in FIG. 5, this goal can be achieved by providing a library of functions which have the same signatures as the calling functions but which call the script translation engine before calling the underlying user action function from the test tool, then:

- either providing modified/renamed 'include' files which reference the script translation enabled library and having the test script reference that library, or
- naming the script translation enabled library with the same name as the test tool library and having the test script reference it (in this case, the script translation enabled library would still reference the original functions).

As shown in the FIG. 5, this process 500 of steps for implementing intercept and execution of a user interface action includes a first step 510 of intercepting the user interface action and identifying (i) the action's control type and (ii) its translatable text. At step 520 look up of the resource identifier is performed. At step 520 look up of the translated text is performed. Finally, at step 540 the user interface action is executed with the translated text.

Look up of the resource identifier uses the control type to allow for the case where the same text string is translated differently depending on context. The lookup can be implemented on the resource files or the compiled application binary files.

Look up of the translated text uses the resource ID. This will be a standard operation and can be executed on the compiled application binary files.

Thus, as shown in FIG. 6, the first line (for example) of script section 210 is performed under control of the script translation engine 420 as follows with respect to translated user dialog 'Anmeldung' 320:

- the script line 'EditSetText ("User name", "John Doe")' with causes the user action 'ID_USER_NAME "User name"' to be looked up (as indicated by arrow 610) in source (English) resources library 430
- the entry (user action 'ID_USER_NAME "Benutzername"') with same resource ID is then looked up (as indicated by the arrow 620) in the target (German) resources library 440
- the user action 'ID_USER_NAME "Benutzername"' is then passed back (as indicated by the arrow 630) to the script 210, where it is associated (as indicated by the arrow 640) with the text entry field 'Benutzername' in the user dialog 320, resulting in successful completion of the source line in the script 210.

Thus, test execution in the modified system may be summarised as follows:

- The Script Translation Engine (STE) 420 uses the control type and text string to identify the resource identifier.
- The STE 420 then finds the equivalent resource in the translated version of the resources.
- The STE 420 substitutes the translated text string into the user action function.
- The original user action function is called with the translated text.

It may be noted that if the translated text cannot be located, the test script will fail as previously described above in relation to FIG. 3. However, in this case the failure is valid, e.g., it could be that the version of the translated application differs from the one the test script was written for.

It will therefore be appreciated that the modified automated test system described above is based on the two important features:

- (1) replacement of the default library of test functions (which simulate user actions) with a script translation engine enabled set of test functions, and
- (2) implementation of the script translation engine such that execution of an unmodified test script which will successfully run against a translated version of the application.

The sample code included in the following description is written in Java and uses features of the Java™ platform to illustrate a practical implementation of the key concepts. However, it will of course be understood that the present invention is not limited to such implementation, and may alternatively be implemented using a different programming language for a different platform.

Sample Test Script

Continuing from the example described above, only a subset of the test functions that would typically be provided are considered below, i.e., a function to simulate a user typing text in an editable text field, and a function to simulate a user clicking a button.

To simplify the example, a set of dummy test functions are provided which only output a text description of what the real test functions would do. These are implemented as static methods in a Java™ class as follows:

```
public class TestMethods
{
    static public void EditSetText(String label, String text)
    {
        System.out.println("Type \"" + text + "\" in the \"" + label +
            "\" text edit field.");
    }

    static public void ButtonClick(String label)
    {
```

```
        System.out.println("Click the button \"" + label + "\".");  
    }  
}
```

5 An extract of a test script which uses these functions could look like this:

```
public class TestScript  
{  
10       public static void main(String[] args)  
      {  
          // test initialisation  
  
          // perform user log on  
15       TestMethods.EditSetText("User name", "John Doe");  
          TestMethods.EditSetText("Password", "password");  
          TestMethods.ButtonClick("Log On");  
  
          // remainder of the test case  
20       }  
      }  
}
```

 This test script is run using this command-line (assuming environment is correctly configured):

```
25       java TestScript
```

 And the following output would be generated:

```
30       Type "John Doe" in the "User name" text edit field.  
      Type "password" in the "Password" text edit field.  
      Click the button "Log On".
```

35 It is clear that the original English control labels are used to identify which editable text fields will have text entered and which button will be clicked. As described earlier this test script will fail when run against a translated version of the product.

Script Translation Enabled Test Methods

40 The original test function library needs to be shadowed by an alternative implementation of the test functions which are enabled for script translation. The preferred way to implement this is as follows:

Prevent the original test library from being loaded during test script execution, e.g., by renaming the library or changing its location.

5 Make the script translation enabled version of the test functions available to be loaded. It is the responsibility of this implementation to ensure that the new functions can be used without any test script modification.

10 Continuing from the example started above, at runtime the test script will use the TestMethods class; to prevent it being loaded the class file is renamed from TestMethods.class to TestMethods.impl.

15 A script translation enabled version of the TestMethods class is provided. This implementation will:

- Load the original implementation so the test functions can be called after script translation, and
- Call the script translation engine to translate any text before calling the test function

20 Loading the original implementation in Java™ requires the use of a class loader. This could be implemented as follows:

import java.io.FileInputStream;

25 public class TestMethodsClassLoader extends ClassLoader

{

 Class loadTestMethods(String name) throws ClassNotFoundException

 {

30 FileInputStream fi = null;

 try

 {

 String path = name.replace('.', '/');

 fi = new FileInputStream(path + ".impl");

35 byte[] classBytes = new byte[fi.available()];

 fi.read(classBytes);

 return defineClass(null, classBytes,

 0, classBytes.length);

 }

40 catch (Exception e)

 {

 // we could not find the class

 // so indicate the problem with an exception

 throw new ClassNotFoundException(name);

BEST AVAILABLE COPY


```
    }  
    finally  
    {  
        if (null != fi)  
5        {  
            try  
            {  
                fi.close();  
            }  
10           catch (Exception e)  
            {  
            }  
        }  
    }  
15 }  
}
```

This implementation of a Java class loader simply loads the renamed file (i.e., 'TestMethods.impl') and then defines a new Java class from the loaded bytes. In the script translation engine enabled implementation of the TestMethods class this class loader is used as follows:

```
// load the script translation engine  
scriptTranslationEngine = new ScriptTranslationEngine();  
25  
  
// use a custom classloader to load the original implementation  
TestMethodsClassLoader classLoader = new TestMethodsClassLoader();  
testMethods = classLoader.loadTestMethods("TestMethods");  
editSetText = testMethods.getMethod("EditSetText", editSetTextArgs);  
30 buttonClick = testMethods.getMethod("ButtonClick", buttonClickArgs);
```

The renamed class file is loaded and references to the original test functions can be created and saved for later use. An instance of the script translation engine is also loaded and saved for later use

The script translation enabled test function would look like this:

```
static public void EditSetText(String label, String text)  
{  
40    try  
    {  
        label = scriptTranslationEngine.getTranslatedText("EDIT",  
                                                            label);  
        Object[] args = new Object[] { label, text };  
    }  
}
```

```
        editSetText.invoke(null, args);  
    }  
    catch (Exception e)  
    {  
5      }  
    }
```

Before calling the original test function the script translation engine is first called to get the translated text.

Script Translation Engine

The script translation engine implements the following features:

- 15 • Load the resource files for the version of the application for which the test script was written and on which the test script is to be executed
- Build a lookup table for resolving resource identifiers from resource strings and control types
- 20 • Perform lookups to get the translated text from a resource string and control type

In Java the resource files are typically implemented as property bundles (text files containing resource identifier and resource values name/value pairs). This implementation below assumes the names of the source (the version for which the test script was written) and target (the version on which the test script will be run) resource files names are specified as parameters to the Java™ runtime library. Other runtime libraries will provide a mechanism to allow configuration settings to be configured from an external source, e.g., '.ini' files.

```
// resource files to be used for this test run  
String sourceFile = System.getProperty("source_file");  
String targetFile = System.getProperty("target_file");  
35  
  
// if they are the same or not specified then no translation is required  
if (sourceFile == null || sourceFile.equals(targetFile)) return;  
  
// load the language files  
40 sourceBundle = ResourceBundle.getBundle(sourceFile);  
targetBundle = ResourceBundle.getBundle(targetFile);
```

The lookup table uses the source resource strings as keys and the resource identifiers as values. In the case where the same string has multiple resource identifiers then all of them will be saved.

```
5    // populate the resource id lookup table with all resource values
    // using the resource strings as the keys and the resource ids as the
    // values. if a resource string has multiple resource ids then all of
    // the values are stored in a Vector.
    for (Enumeration e = sourceBundle.getKeys(); e.hasMoreElements();)
10   {
        String key = e.nextElement().toString();
        String val = sourceBundle.getString(key.toString());
        if (resourceIdLookup.containsKey(val))
        {
15         Object keyObject = resourceIdLookup.get(val);
            Vector vals = null;
            if (keyObject instanceof Vector)
            {
                vals = (Vector) keyObject;
20         }
            else
            {
                vals = new Vector();
                vals.add(keyObject);
25         }
            vals.add(key);
            resourceIdLookup.put(val, vals);
        }
        else
30     {
            resourceIdLookup.put(val, key);
        }
    }
```

Lookups require the control type and also the resource string to be specified. The example below demonstrates one strategy for resolving a resource ID where there are multiple resource IDs for a single string. This is important because in the translated version of the resource file the string could be translated in two different ways depending on context.

```
40    String getTranslatedText(String control, String text)

    {
        if (resourceIdLookup.containsKey(text))
        {
```

```
Object key = resourceIdLookup.get(text);
if (key instanceof Vector)
{
    // simple strategy for finding the correct resource id
    // is to assume the id starts with the control type
    for (Enumeration e = ((Vector)key).elements();
        e.hasMoreElements(); )
    {
        String nextKey = e.nextElement().toString();
        if (nextKey.toString().startsWith(control))
        {
            text = targetBundle.getString(nextKey);
            break;
        }
    }
}
else
{
    text = targetBundle.getString(key.toString());
}
return text;
}
```

Sample Translated Test Script

To run the previous test script with script translation enabled the following command-line is used (assume environment is correctly configured):

```
java -Dsource_file=resources_en -Dtarget_file=resources_de TestScript
```

The following files need to be available on the local file system:

- (1) 'resources_en.properties' - The English version of the application resources
- (2) 'resources_de.properties' - The German version of the application resources
- (3) 'TestMethods.impl' - The original implementation of the test functions

And the following output would be generated:

Type "John Doe" in the "Benutzername" text edit field.

BEST AVAILABLE COPY

Type "password" in the "Kennwort" text edit field.
Click the button "Anmeldung".

Thus, it will be understood that the novel scheme described above provides a method for overriding relevant Automation Script APIs (Application Programming Interfaces) which is easy to install/uninstall/integrate with an automation test tool, which allows Fast Resource lookup, 'write once read many' functionality, and easy subclassing of automation functions. It allows interception of selected runtime methods (of the automation environment) to perform translation on the fly of their parameters and subsequent calling of the native, unmodified automation environment methods with the translated parameters. Also, it permits runtime lookup of the original tested application translated resources, i.e., the same ones used by the tested application. It also allows fast resource lookup, resource lookup across multiple resource files in multiple directories, and resource lookup for partial strings. Also, it readily permits a log file of activity to be created.

It will be understood that the system and method described above provides language-neutral runtime user interface automation offering the following advantages:

- Run-time mapping of the user interface components of the automation script using the original English automation script as the reference. This has great potential because it does not attach any extra burden on the script developer and requires little extra from the user - other than providing the English reference file.
- Non interception methodology when compared with the prior art U.S. patent no. 5,475,843.
- Speed - as the methodology in this idea incorporates a non-interference directive the application is allowed to execute in an unhindered way. It is common knowledge that interception based methodologies have the side effect of slowing down the running application.
- Less complexity for Script Writer, i.e., the script writer can write the automation script and in so doing is not concerned with designing the script to accommodate language/translated characteristics.
- Supports a level of cooperation from the application with regard to proprietary custom controls, i.e., respects proprietary controls and provides an accessibility methodology based on what the application exposes by way of its application programming interface (API) and user interface (UI).

- Is automation centric in that the automation is seen as the driver of the application's natural execution. In the methodology of the prior art U.S. patent no. 5,475,843 the application is centric, in that the interception of all applications messages, events (keyboard, mouse, system), states, etc., is a pre-requisite to success. This prior art methodology can be considered a flawed model as once again it interferes with the natural execution of the application. This gives rise to quality and engineering considerations which can undermine the true value of the automation being run.
- The methodology used in the prior art U.S. patent no. 5,475,843 encourages a level of abstraction which can potentially take some degree of control from the script writer. In this regard the script writer is somewhat dependent on the methodology and characteristics of the prior art patent's interception and event handling engine. This limits the degree of control that the script writer can have in the prior art patent's natural test abstraction development methodology. The novel scheme described above assumes a non-interference directive, and therefore looks to a cooperating application to achieve the test automation goals. This is can be seen as a key difference: the novel scheme described above supports non-interference, while the prior art patent relies on interference. In summary, the prior art patent's solution solves a similar class of problem to that of the novel scheme described above, but in a way which encourages interference and interception of the running application. In contrast, the novel scheme described above protects the application's natural run-time execution, i.e., during the application's run time a language setting decides what language is being tested, and the script writer's additional effort is minimised in supporting the run-time testing of this localised application. Also, the novel scheme described above simply takes the localised components of existing translated applications and seamlessly makes these available during the script's run time - therefore not depending on a database. The novel scheme described above may be considered as effectively subclassing the script to allow a language-neutral execution of the script. This differs to the prior art patent's solution which relies on the specific interception of the running application to achieve its goals. Therefore the need to write ATUs (as seen in the prior art patent) is unnecessary in the methodology of the novel scheme described above.
- Test script is written once and re-used across all languages.
- No need to translate the tests script to run on translated products or applications.

- No test script modifications required when running on different language versions
- Solution is independent from the target control type implementations (i.e., the automation environment is left unmodified to deal with those aspects), platforms, technologies, etc.
- Can be applied to various automation environments independently of them.
- No additional effect on the runtime environment of the tested application compared to the same automation running without the novel scheme described above.

It will be appreciated that the method described above for language-neutral automation of test scripts will typically be carried out in software running on a processor (not shown) within the system, and that the software may be provided as a computer program element carried on any suitable data carrier (also not shown) such as a magnetic or optical computer disc.

It will be understood that appreciated that alternatives to the aspects of the novel scheme described above will be apparent to a person of ordinary skill in the art, without departing from the underlying inventive concept. For example, although the novel scheme has been described in the context of a graphical user interface, it will be appreciated that the invention may be employed in the context an alternative user interface such as a text user interface.

CLAIMS

1. A system for runtime user interface automation, comprising:

5 automation script means for receiving an automation script for automating use of the user interface in an application; and

script translation means for intercepting a call from the automation script to a function simulating a user action on the application,
10 retrieving a translated text string associated with the function call, and calling the function simulating the user action with the translated text string.

2. The system of claim 1 wherein the script translation means comprises:

15 message translation means for supplying translated text for the automation script's run time execution; and

selective text locator means coupled to the message translation means
20 for selectively supplying appropriate text to the automation script's run time execution depending on the function call.

3. The system of claim 2 wherein the selective text locator means is arranged to selectively supply appropriate text to the automation script's
25 run time execution depending on a resource ID of the function call.

4. The system of claim 1, 2 or 3 wherein the script translation means comprises:

30 a library including a function having a same signature as the function call and which is arranged to retrieve the translated text string before the function call; and

one of A-B:

35 A a file referencing the library, the automation script being arranged to reference the file and the library, and

B the library including the retrieval function and the function call, the library being arranged to be called by the automation script.

40 5. The system of claim 4 wherein the file referencing the library comprises an include file.

BEST AVAILABLE COPY

6. The system of claim 4 wherein the library including the retrieval function and the function call has the same name as a library containing the function called by the automation script.

5 7. The system of any preceding claim wherein the automation script comprises a Java™ script.

8. The system of any preceding claim wherein the automation script is in the English language and the application is arranged to use a non-English language.

9. The system of any preceding claim wherein the user interface comprises a graphical user interface.

10 10. A method for runtime user interface automation, comprising:

providing an automation script for automating use of the user interface in an application;

20 intercepting a call from the automation script to a function simulating a user action on the application;

retrieving a translated text string associated with the function call; and

25 calling the function simulating the user action with the translated text string.

11. The method of claim 10 further comprising:

30 providing message translation means for supplying translated text for the automation script's run time execution; and

35 providing selective text locator means coupled to the message translation means,

wherein the step of retrieving comprises selectively supplying appropriate text by the selective text locator means to the automation script's run time execution depending on the function call.

40 12. The method of claim 11 wherein the selective text locator means selectively supplies appropriate text to the automation script's run time execution depending on a resource ID of the function call.

13. The method of claim 10, 11 or 12 wherein the script translation means comprises:

5 a library including a function having a same signature as the function call and which is called to retrieve the translated text string before the function call; and

one of A-B:

10 A a file referencing the library, the automation script referencing the file and the library, and

15 B the library including the retrieval function and the function call, the library being called by the automation script.

14. The method of claim 13 wherein the file referencing the library comprises an include file.

20 15. The method of claim 13 wherein the library including the retrieval function and the function call has the same name as a library containing the function called by the automation script.

25 16. The method of any one of claims 10-15 wherein the automation script comprises a Java™ script.

17. The method of any one of claims 10-16 wherein the automation script uses the English language and the application uses a non-English language.

30 18. The method of any one of claims 10-17 wherein the user interface comprises a graphical user interface.

19. A computer program element comprising computer program means for performing substantially the method of any one of claims 10-18.

35 20. A system for runtime user interface automation substantially as hereinbefore described with reference to FIGS. 4-6 of the accompanying drawings.

40 21. A method for runtime user interface automation substantially as hereinbefore described with reference to FIGS. 4-6 of the accompanying drawings.

ABSTRACT

SYSTEM AND METHOD FOR USER INTERFACE AUTOMATION

5 A system and method for language-neutral runtime user interface
automation by: providing an automation script (410) for automating use of
the user interface in an application; intercepting (510) a call from the
automation script to a function simulating a user action on the
10 application; retrieving (530) a translated text string associated with the
function call; and calling (540) the function simulating the user action
with the translated text string.

15 The scheme provides a method for overriding relevant automation
script APIs, is easy to install/uninstall/integrate with an automation test
tool, allows fast resource lookup, and easy subclassifying of automation
functions.

20 This provides the advantages of not attaching any extra burden on the
script developer and requiring little extra from the user (other than
providing a native-language reference file). The scheme is automation
centric and allows the application to execute otherwise unhindered,
allowing it to be applied to various automation environments.

BEST AVAILABLE COPY

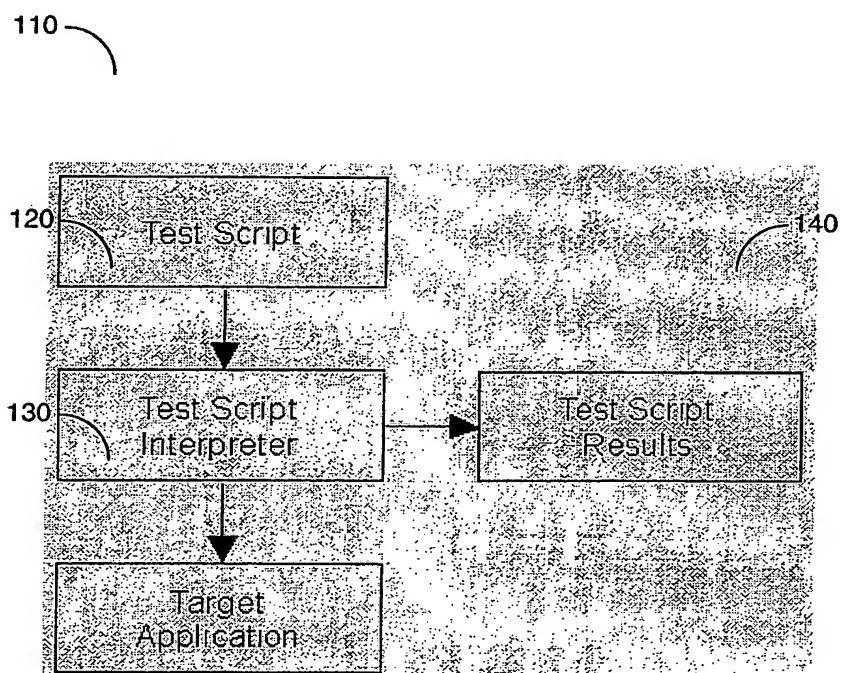


FIG. 1
Prior Art

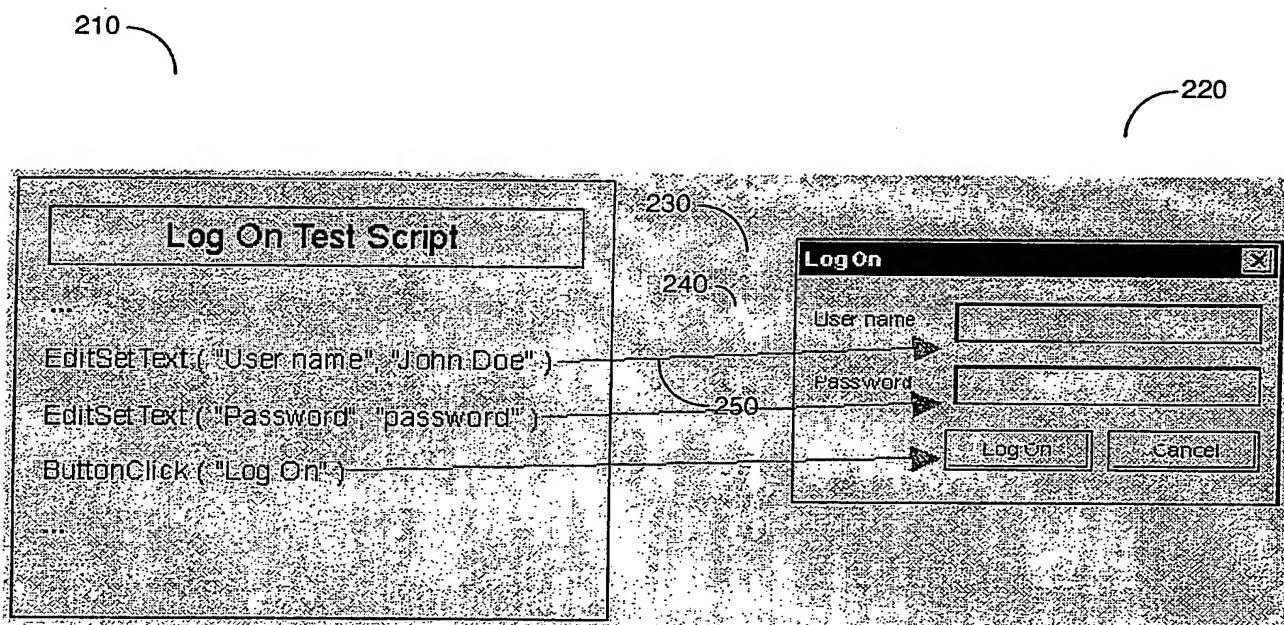


FIG. 2
Prior Art

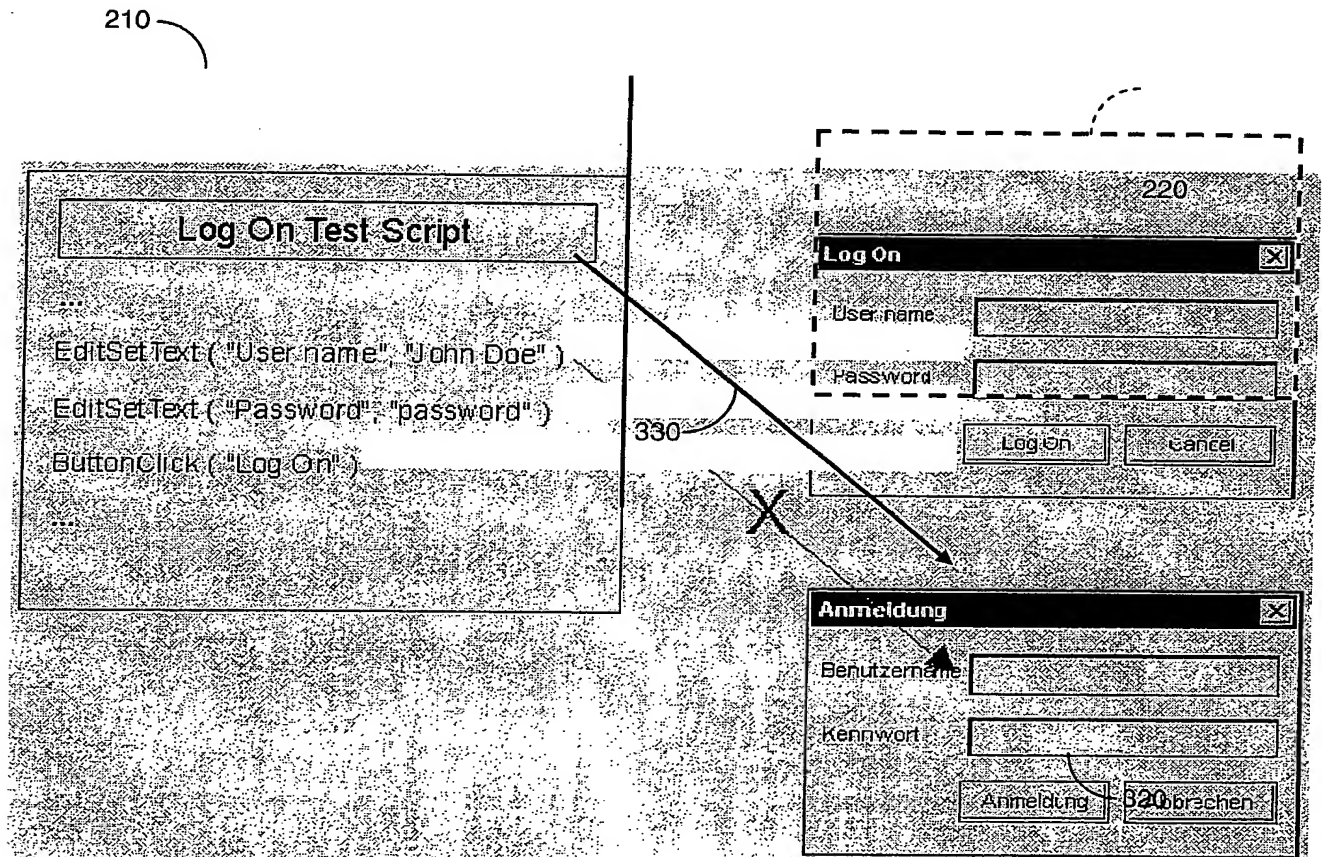


FIG. 3
Prior Art

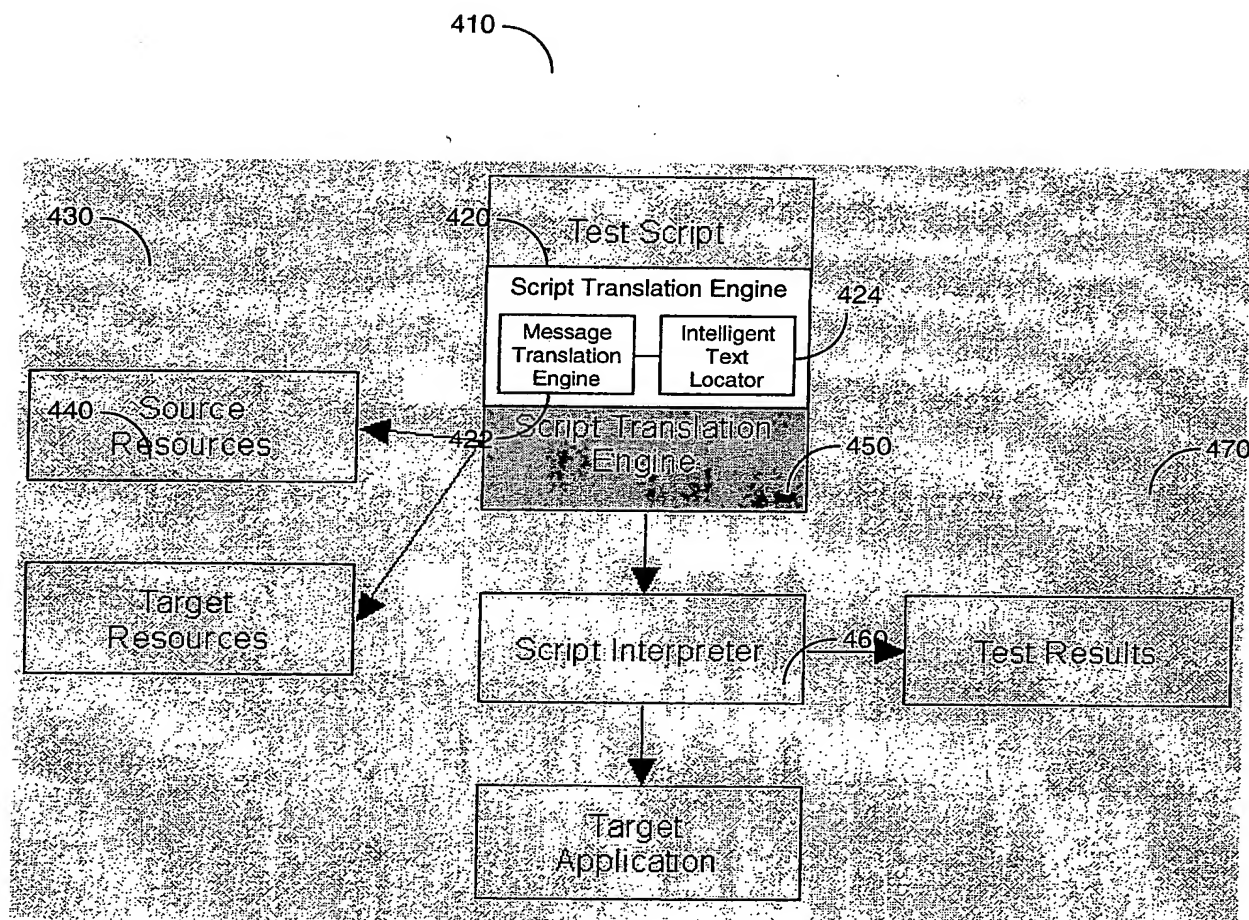
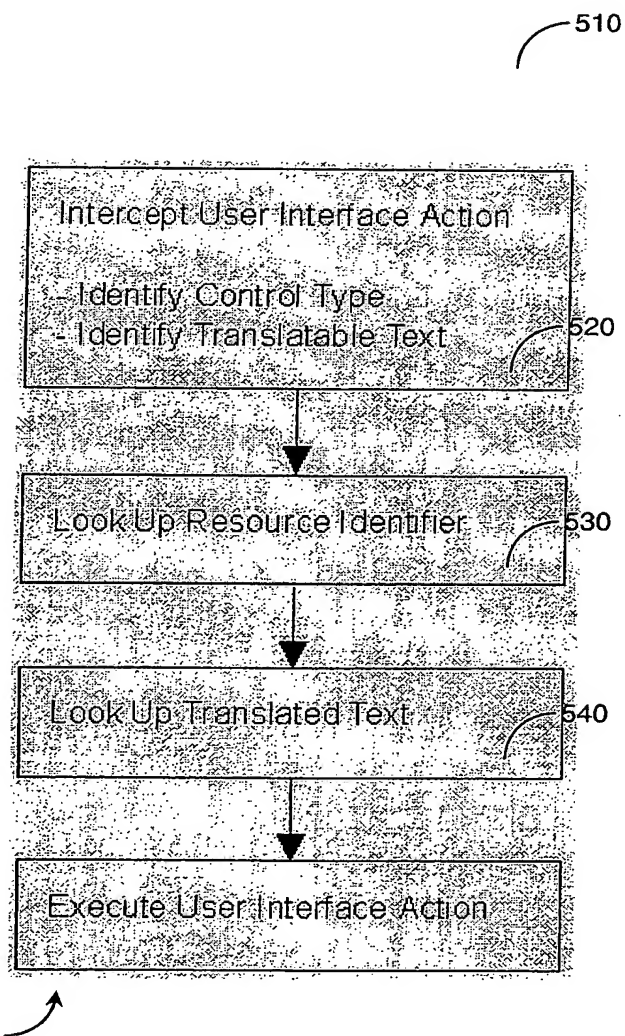


FIG. 4

400

BEST AVAILABLE COPY

**FIG. 5**

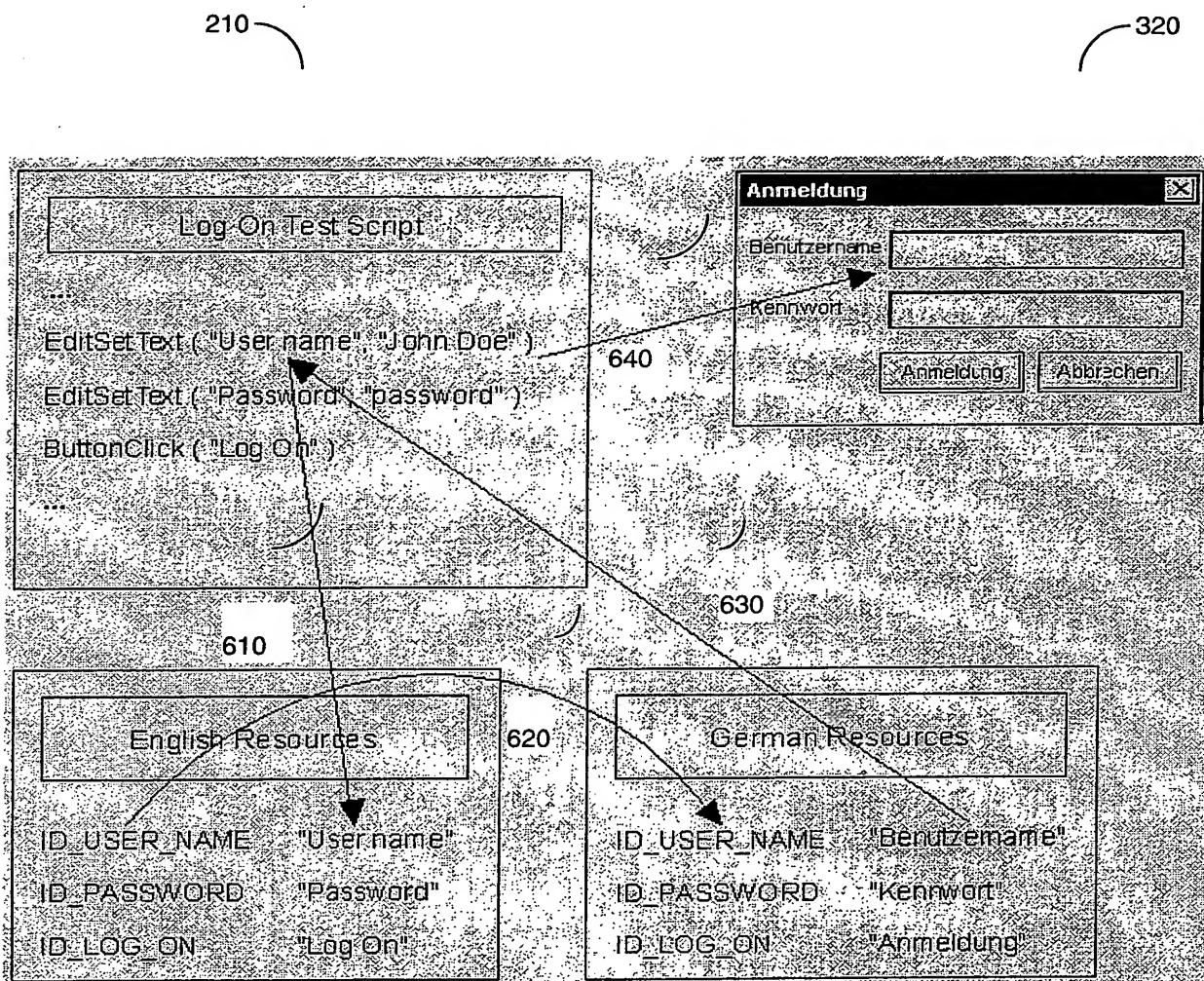


FIG. 6

BEST AVAILABLE COPY